Package ‘dbscan’

December 15, 2015

Version 0.9-6
Date 2015-12-14
Title Density Based Clustering of Applications with Noise (DBSCAN) and Related Algorithms
Description A fast reimplementation of several density-based algorithms of the DBSCAN family for spatial data. Includes the DBSCAN (density-based spatial clustering of applications with noise) and OPTICS (ordering points to identify the clustering structure) clustering algorithms and the LOF (local outlier factor) algorithm. The implementations uses the kd-tree data structure (from library ANN) for faster k-nearest neighbor search. An R interface to fast kNN and fixed-radius NN search is also provided.
Imports Rcpp, graphics, stats, methods
LinkingTo Rcpp
Suggests fpc, microbenchmark, testthat
BugReports https://github.com/mhahsler/dbscan/issues
License GPL (>= 2)
Copyright ANN library is copyright by University of Maryland, Sunil Arya and David Mount. All other code is copyright by Michael Hahsler.
NeedsCompilation yes
Author Michael Hahsler [aut, cre, cph], Sunil Arya [ctb, cph], David Mount [ctb, cph]
Maintainer Michael Hahsler <mhahsler@lyle.smu.edu>
Repository CRAN
Date/Publication 2015-12-15 01:14:05

R topics documented:

dbscan ......................................................... 2
frNN .......................................................... 4
kNN ............................................................ 6
Description

Fast reimplementation of the DBSCAN (Density-based spatial clustering of applications with noise) clustering algorithm using a kd-tree. The implementation is significantly faster and can work with larger data sets than \texttt{dbscan} in \texttt{fpc}.

Usage

\begin{verbatim}
dbscan(x, eps, minPts = 5, weights = NULL,
borderPoints = TRUE, search = "kdtree", bucketSize = 10,
splitRule = "suggest", approx = 0)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{x} a data matrix or a dist object.
  \item \texttt{eps} size of the epsilon neighborhood.
  \item \texttt{minPts} number of minimum points in the eps region (for core points). Default is 5 points.
  \item \texttt{weights} numeric; weights for the data points. Only needed to perform weighted clustering.
  \item \texttt{borderPoints} logical; should border points be assigned. The default is \texttt{TRUE} for regular DBSCAN. If \texttt{FALSE} then border points are considered noise (see DBSCAN* in Campello et al, 2013).
  \item \texttt{search} nearest neighbor search strategy (one of "kdtree" or "linear", "dist").
  \item \texttt{bucketSize} max size of the kd-tree leafs.
  \item \texttt{splitRule} rule to split the kd-tree. One of "STD", "MIDPT", "FAIR", "SL\_MIDPT", "SL\_FAIR" or "SUGGEST" (SL stands for sliding). "SUGGEST" uses ANNs best guess.
  \item \texttt{approx} relative error bound for approximate nearest neighbor searching.
\end{itemize}

Details

This implementation of DBSCAN implements the original algorithm as described by Ester et al (1996). DBSCAN estimates the density around each data point by counting the number of points in a user-specified eps-neighborhood and applies a used-specified minPts thresholds to identify core, border and noise points. In a second step, core points are joined into a cluster if they are density-reachable (i.e., there is a chain of core points where one falls inside the eps-neighborhood of the
Border points are arbitrarily assigned to clusters in the original algorithm. DBSCAN* (see Campello et al 2013) treats all border points as noise points. This is implemented with borderPoints = FALSE.

Setting parameters for DBSCAN: \texttt{minPts} is often set to be dimensionality of the data plus one. The knee in \texttt{kNndistplot} can be used to find suitable values for \texttt{eps}.

See \texttt{kNN} for more information on the other parameters related to nearest neighbor search.

\textbf{Value}

A object of class 'dbscan' with the following components:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{eps}</td>
<td>value of the \texttt{eps} parameter.</td>
</tr>
<tr>
<td>\texttt{minPts}</td>
<td>value of the \texttt{minPts} parameter.</td>
</tr>
<tr>
<td>\texttt{cluster}</td>
<td>A integer vector with cluster assignments. Zero indicates noise points.</td>
</tr>
</tbody>
</table>

\textbf{Author(s)}

Michael Hahsler

\textbf{References}

Martin Ester, Hans-Peter Kriegel, Joerg Sander, Xiaowei Xu (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Institute for Computer Science, University of Munich. \textit{Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)}.


\textbf{See Also}

\texttt{kNndistplot, dbscan} in \texttt{fpc}.

\textbf{Examples}

```r
   data(iris)
   iris <- as.matrix(iris[,1:4])

   res <- dbscan(iris, eps = .4, minPts = 4)
   res

   pairs(iris, col = res$cluster + 1L)

   ## compare with dbscan from package fpc (only if installed)
   if (requireNamespace("fpc", quietly = TRUE)) {
     res2 <- fpc::dbscan(iris, eps = .4, MinPts = 4)
     pairs(iris, col = res2$cluster + 1L)
   }
```
## Make sure both versions produce the same results

```r
all(res$cluster == res2$cluster)
```

## Find suitable eps parameter (look at knee)

```r
kNNdistplot(iris, k = 4)
```

## Example data from fpc

```r
set.seed(665544)
n <- 100
x <- cbind(
  x = runif(10, 0, 10) + rnorm(n, sd = 0.2),
  y = runif(10, 0, 10) + rnorm(n, sd = 0.2)
)
res <- dbscan::dbscan(x, eps = .2, minPts = 4)
res
```

```r
plot(x, col = res$cluster + 1L)
```

## Not run:

### Compare speed against fpc version (if microbenchmark is installed)

```r
if (requireNamespace("microbenchmark", quietly = TRUE)) {
  t_dbscan <- microbenchmark::microbenchmark(
    dbscan::dbscan(x, .2, 4), times = 10, unit = "ms"
  )
  t_dbscan_linear <- microbenchmark::microbenchmark(
    dbscan::dbscan(x, .2, 4, search = "linear"), times = 10, unit = "ms"
  )
  t_fpc <- microbenchmark::microbenchmark(
    fpc::dbscan(x, .2, 4), times = 10, unit = "ms"
  )
  rbind(t_fpc, t_dbscan_linear, t_dbscan)
  boxplot(rbind(t_fpc, t_dbscan_linear, t_dbscan),
    names = c("fpc (R)", "dbscan (linear)", "dbscan (kdtree)"),
    main = "Runtime comparison in ms")
}

## End (not run)

---

**frNN**

*Find the Fixed Radius Nearest Neighbors*

## Description

This function uses a kd-tree to find the fixed radius nearest neighbors (including distances) fast.
Usage

\texttt{frNN(x, eps, sort = TRUE, search = "kdtree", bucketSize = 10, splitRule = "suggest", approx = 0)}

Arguments

\texttt{x} \hspace{1cm} \text{a data matrix or a dist object.}
\texttt{eps} \hspace{1cm} \text{neighbors radius.}
\texttt{search} \hspace{1cm} \text{nearest neighbor search strategy (one of "kdtree" or "linear", "dist").}
\texttt{sort} \hspace{1cm} \text{sort the neighbors by distance?}
\texttt{bucketSize} \hspace{1cm} \text{max size of the kd-tree leafs.}
\texttt{splitRule} \hspace{1cm} \text{rule to split the kd-tree. One of "STD", "MIDPT", "FAIR", "SL_MIDPT", "SL_FAIR" or "SUGGEST" (SL stands for sliding). "SUGGEST" uses ANNs best guess.}
\texttt{approx} \hspace{1cm} \text{use approximate nearest neighbors. All NN up to a distance of a factor of 1+approx eps may be used. Some actual NN may be omitted leading to spurious clusters and noise points. However, the algorithm will enjoy a significant speedup.}

Details

For details on the parameters see \texttt{kNN}.

Note: self-matches are not returned!

Value

A list with the following components:

\texttt{dist} \hspace{1cm} \text{a matrix with distances.}
\texttt{id} \hspace{1cm} \text{a matrix with ids.}
\texttt{eps} \hspace{1cm} \text{eps used.}

Author(s)

Michael Hahsler

References


See Also

\texttt{kNN} for k nearest neighbor search.
Examples

data(iris)

# Find fixed radius nearest neighbors for each point
nn <- frNN(iris[,-5], eps=.5)

# Number of neighbors
hist(sapply(nn$id, length),
     xlab = "k", main="Number of Neighbors",
     sub = paste("Neighborhood size eps =", nn$eps))

# Explore neighbors of point i = 10
i <- 10
nn$id[i]
nn$dist[i]

plot(iris[,-5], col = ifelse(1:nrow(iris) %in% nn$id[i], "red", "black"))

---

kNN

**Find the k Nearest Neighbors**

Description

This function uses a kd-tree to find all k nearest neighbors in a data matrix (including distances) fast.

Usage

```r
kNN(x, k, sort = TRUE, search = "kdtree", bucketSize = 10,
    splitRule = "suggest", approx = 0)
```

Arguments

- **x**: a data matrix or a dist object.
- **k**: number of neighbors to find.
- **search**: nearest neighbor search strategy (one of "kdtree", "linear" or "dist").
- **sort**: sort the neighbors by distance?
- **bucketSize**: max size of the kd-tree leafs.
- **splitRule**: rule to split the kd-tree. One of "STD", "MIDPT", "FAIR", "SL_MIDPT", "SL_FAIR" or "SUGGEST" (SL stands for sliding). "SUGGEST" uses ANNs best guess.
- **approx**: use approximate nearest neighbors. All NN up to a distance of a factor of 1+approx eps may be used. Some actual NN may be omitted leading to spurious clusters and noise points. However, the algorithm will enjoy a significant speedup.
Details

If \( x \) is a matrix then this implementation uses the ANN library (see Mount and Arya, 2010) for nearest neighbor search. Linear nearest neighbor search can be used. To speed up nearest neighbor search the kd-tree is used.

Note: self-matches are removed!

bucketsize and splitRule influence how the kd-tree is built. approx uses the approximate nearest neighbor search implemented in ANN. All nearest neighbors up to a distance of \( \text{eps}/(1+\text{approx}) \) will be considered and all with a distance greater than \( \text{eps} \) will not be considered. The other points might be considered. Note that this results in some actual nearest neighbors being omitted leading to spurious clusters and noise points. However, the algorithm will enjoy a significant speedup. For more details see Mount and Arya (2010).

Value

A list with the following components:

- \text{dist} \hspace{1cm} a matrix with distances.
- \text{id} \hspace{1cm} a matrix with ids.
- \text{k} \hspace{1cm} number of \( k \) used.

Author(s)

Michael Hahsler

References


See Also

frNN for fixed radius nearest neighbors.

Examples

data(iris)
\# finding kNN directly in data (using a kd-tree)
nn <- kNN(iris[, -5], k=10)
nn
\# explore neighborhood of point 10
i <- 10
nn$id[i,]
plot(iris[, -5], col = ifelse(1:nrow(iris) %in% nn$id[i,], "red", "black"))
**kNNdist**

*Calculate and plot the k-Nearest Neighbor Distance*

**Description**

Fast calculation of the k-nearest neighbor distances in a matrix of points. The plot can be used to help find a suitable value for the eps neighborhood for DBSCAN. Look for the knee in the plot.

**Usage**

```r
kNNdist(x, k, ...)
kNNdistplot(x, k = 4, ...)
```

**Arguments**

- `x`: the data set as a matrix or a dist object.
- `k`: number of nearest neighbors used (use minPoints).
- `...`: further arguments are passed on to `knn`.

**Details**

See `kNN` for a discussion of the kd-tree related parameters.

**Value**

`kNNdist` returns a numeric vector with the distance to its k nearest neighbor.

**Author(s)**

Michael Hahsler

**See Also**

`kNN`.

**Examples**

```r
data(iris)
iris <- as.matrix(iris[,1:4])

kNNdist(iris, k=4, search="kd")
kNNdistplot(iris, k=4)
# the knee is around a distance of .5

cl <- dbscan(iris, eps = .5, minPts = 4)
pairs(iris, col = cl$cluster+1L)
# Note: black are noise points
```
**lof**  
*Local Outlier Factor Score*

**Description**

Calculate the Local Outlier Factor (LOF) score for each data point using a kd-tree to speed up kNN search.

**Usage**

`lof(x, k = 4, ...)`

**Arguments**

- `x`: a data matrix or a dist object.
- `k`: size if neighborhood.
- `...`: further arguments are passed on to `knn`.

**Details**

LOF compares the local density of an point to the local densities of its neighbors. Points that have a substantially lower density than their neighbors are considered outliers. A LOF score of approximately 1 indicates that density around the point is comparable to its neighbors. Scores significantly larger than 1 indicate outliers.

**Value**

A numeric vector of length `ncol(x)` containing LOF values for all data points.

**Author(s)**

Michael Hahsler

**References**


**See Also**

`knn`
Examples

```r
set.seed(665544)
n <- 100
x <- cbind(
  x=runif(10, 0, 5) + rnorm(n, sd=0.4),
  y=runif(10, 0, 5) + rnorm(n, sd=0.4)
)

### calculate LOF score
lof <- lof(x, k=4)

### distribution of outlier factors
summary(lof)
hist(lof, breaks=10)

### point size is proportional to LOF
plot(x, pch = ".", main = "LOF (k=4)")
points(x, cex = (lof-1)*3, pch = 1, col="red")
text(x[lof>2,], labels = round(lof, 1)[lof>2], pos = 1)
```

optics

**OPTICS**

Description

Implementation of the OPTICS (Ordering points to identify the clustering structure) clustering algorithm using a kd-tree.

Usage

```r
optics(x, eps, minPts = 5, eps_cl,
  search = "kdtree", bucketSize = Inf,
  splitRule = "suggest", approx = 0)
```

Arguments

- **x** a data matrix or a distance matrix.
- **eps** upper limit of the size of the epsilon neighborhood.
- **minPts** number of minimum points in the eps region (for core points). Default is 5 points.
- **eps_cl** Threshold to identify clusters (eps_cl <= eps).
- **search** nearest neighbor search strategy (one of "kdtree", "linear" search, or precomputed "dist") Using precomputed distances is better for high-dimensional, but smaller data sets.
- **bucketSize** max size of the kd-tree leaves.
splitRule  rule to split the kd-tree. One of "STD", "MIDPT", "FAIR", "SL_MIDPT", "SL_FAIR" or "SUGGEST" (SL stands for sliding). "SUGGEST" uses ANNs best guess.

approx  relative error bound for approximate nearest neighbor searching.

Details
This implementation of OPTICS implements the original algorithm as described by Ankers et al (1999). OPTICS is similar to DBSCAN, however, for OPTICS eps is only an upper limit for the neighborhood size used to reduce computational complexity. Similar to DBSCAN, minPts is often set to be dimensionality of the data plus one.

OPTICS linearly orders the data points such that points which are spatially closest become neighbors in the ordering. The closest analog to this ordering is dendrogram in single-link hierarchical clustering. The algorithm also calculates the reachability distance for each point. plot() produces a reachability-plot which shows each points reachability distance where the points are sorted by OPTICS. Valleys represent clusters (the deeper the valley, the more dense the cluster) and high points indicate points between clusters.

If eps.cl is specified, then an algorithm to extract clusters (see Ankers et al, 1999) is used. That is, it internally calls optics_cut to extract the clustering. The resulting clustering is similar to what DBSCAN would produce. The only difference is that OPTICS is not able to assign some border points and reports them instead as noise.

See knn for more information on the other parameters related to nearest neighbor search.

Value
An object of class 'optics' with components:

eps  value of eps parameter.

minPts  value of minPts parameter.

order  optics order for the data points in x.

reachdist  reachability distance for each data point in x.

coredist  core distance for each data point in x.

If eps.cl was specified or optics_cut was called, then in addition the following components are available:

eps.cl  reachability distance for each point in x.

cluster  assigned cluster labels in the order of the data points in x.

Author(s)
Michael Hahsler

References
See Also
dbscan in fpc.

Examples

```r
set.seed(2)
n <- 400

x <- cbind(
  x = runif(4, 0, 1) + rnorm(n, sd=0.1),
  y = runif(4, 0, 1) + rnorm(n, sd=0.1)
)

plot(x, col=rep(1:4, time = 100))

### run OPTICS
res <- optics(x, eps = 1, minPts = 10)
res

### get order
res$order

### plot produces a reachability plot
plot(res)

### identify clusters by cutting the reachability plot (black is noise)
res <- optics_cut(res, eps_cl = .065)
res

plot(res)
plot(x, col = res$cluster+1L)

### re-cutting at a higher eps threshold
res <- optics_cut(res, eps_cl = .1)
res
plot(res)
plot(x, col = res$cluster+1L)

### using OPTICS on a precomputed distance matrix
d <- dist(x)
res <- optics(x, eps = 1, minPts = 10)
plot(res)
```
Index

+Topic clustering
dbscan, 2
optics, 10
+Topic model
dbscan, 2
frNN, 4
kNN, 6
kNNdist, 8
lof, 9
optics, 10
+Topic plot
kNNdist, 8

DBSCAN (dbscan), 2
dbscan, 2, 3, 12
frNN, 4, 7
frnn (frNN), 4

kNN, 3, 5, 6, 8, 9, 11
knn (kNN), 6
kNNdist, 8
kNNdistplot, 3
kNNdistplot (kNNdist), 8

LOF (lof), 9
lof, 9

OPTICS (optics), 10
optics, 10
optics_cut (optics), 10